자료구조 Homework #9

충북대학교 컴퓨터공학과 2022040014 오재식

```
/*
```

```c
 * Binary Search Tree #1
 *
 * Data Structures
 *
 * School of Computer Science
 * at Chungbuk National University
 *
 */

#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int key;
    struct node *left;
    struct node *right;
} Node;

int initializeBST(Node** h);

/* functions that you have to implement */
void inorderTraversal(Node* ptr);     /* recursive inorder traversal */
void preorderTraversal(Node* ptr);    /* recursive preorder traversal */
void postorderTraversal(Node* ptr);   /* recursive postorder traversal */
int insert(Node* head, int key);  /* insert a node to the tree */
int deleteLeafNode(Node* head, int key);  /* delete the leaf node for the key */
Node* searchRecursive(Node* ptr, int key);  /* search the node for the key */
Node* searchIterative(Node* head, int key);  /* search the node for the key */
int freeBST(Node* head); /* free all memories allocated to the tree */

/* you may add your own defined functions if necessary */


int main()
{
    char command;
    int key;
    Node* head = NULL;
    Node* ptr = NULL;   /* temp */
    printf("------- ohjaesik ---- 2022040014 ------")
    do{
        printf("\n\n");
        printf("----------------------------------------------------------------
\n");
        printf("                   Binary Search Tree #1
\n");
        printf("----------------------------------------------------------------
\n");
        printf(" Initialize BST       = z
\n");
```

```
        printf(" Insert Node          = n      Delete Leaf Node             = d
\n");
        printf(" Inorder Traversal    = i      Search Node Recursively      = s
\n");
        printf(" Preorder Traversal   = p      Search Node Iteratively      =
f\n");
        printf(" Postorder Traversal  = t      Quit                         =
q\n");
        printf("--------------------------------------------------------------
\n");

        printf("Command = ");
        scanf(" %c", &command);

        switch(command) {
        case 'z': case 'Z':
            initializeBST(&head);
            break;
        case 'q': case 'Q':
            freeBST(head);
            break;
        case 'n': case 'N':
            printf("Your Key = ");
            scanf("%d", &key);
            insert(head, key);
            break;
        case 'd': case 'D':
            printf("Your Key = ");
            scanf("%d", &key);
            deleteLeafNode(head, key);
            break;
        case 'f': case 'F':
            printf("Your Key = ");
            scanf("%d", &key);
            ptr = searchIterative(head, key);
            if(ptr != NULL)
                printf("\n node [%d] found at %p\n", ptr->key, ptr);
            else
                printf("\n Cannot find the node [%d]\n", key);
            break;
        case 's': case 'S':
            printf("Your Key = ");
            scanf("%d", &key);
            ptr = searchRecursive(head->left, key);
            if(ptr != NULL)
                printf("\n node [%d] found at %p\n", ptr->key, ptr);
            else
                printf("\n Cannot find the node [%d]\n", key);
            break;
```

```
        case 'i': case 'I':
            inorderTraversal(head->left);
            break;
        case 'p': case 'P':
            preorderTraversal(head->left);
            break;
        case 't': case 'T':
            postorderTraversal(head->left);
            break;
        default:
            printf("\n       >>>>>   Concentration!!   <<<<<     \n");
            break;
        }

    }while(command != 'q' && command != 'Q');

    return 1;
}

int initializeBST(Node** h) {

    /* if the tree is not empty, then remove all allocated nodes from the tree*/
    if(*h != NULL)
        freeBST(*h);

    /* create a head node */
    *h = (Node*)malloc(sizeof(Node));
    (*h)->left = NULL;  /* root */
    (*h)->right = *h;
    (*h)->key = -9999;
    return 1;
}



void inorderTraversal(Node* ptr) //중위 순회로 출력.
{
    if(ptr) {
        inorderTraversal(ptr->left); // 왼쪽 노드부터 순회
        printf(" [%d] ", ptr->key); // ptr 의 key 출력
        inorderTraversal(ptr->right); // 오른쪽 노드 순회
    }
}

void preorderTraversal(Node* ptr) //전위 순회로 출력.
{
    if(ptr) {
        printf(" [%d] ", ptr->key); // ptr 의 값 출력.
        preorderTraversal(ptr->left); // 왼쪽노드로 이동
```

```
        preorderTraversal(ptr->right); // 오른쪽노드로 이동
    }
}

void postorderTraversal(Node* ptr)
{
    if(ptr) {
        postorderTraversal(ptr->left); //왼쪽노드로 이동
        postorderTraversal(ptr->right); // 오른쪽노드로 이동
        printf(" [%d] ", ptr->key); // ptr 의 값 출력
    }
}


int insert(Node* head, int key)
{
    Node* newNode = (Node*)malloc(sizeof(Node)); // 메모리 공간 할당
    newNode->key = key; // key 값 할당
    newNode->left = NULL;
    newNode->right = NULL;

    if (head->left == NULL) { //head 의 노드가 비어있다면
        head->left = newNode; // head 의 왼쪽에 새로운 노드 삽입
        return 1;
    }

    /* head->left is the root */
    Node* ptr = head->left;

    Node* parentNode = NULL;
    while(ptr != NULL) {

        /* if there is a node for the key, then just return */
        if(ptr->key == key) return 1; //만약에 이미 key 값을 트리가 가지고 있다면 return

        /* we have to move onto children nodes,
         * keep tracking the parent using parentNode */
        parentNode = ptr;

        /* key comparison, if current node's key is greater than input key
         * then the new node has to be inserted into the right subtree;
         * otherwise the left subtree.
         */
        if(ptr->key < key) //현재가리키고 있는 노드의 key 값보다 삽입된 key 가 크다면
            ptr = ptr->right; // 오른쪽 자식노드로 이동
        else
            ptr = ptr->left; // 왼쪽 자식노드로 이동
    }
```

```
    /* linking the new node to the parent */
    if(parentNode->key > key) //삽입될 노드가 부모노드보다 작다면 왼쪽
        parentNode->left = newNode;
    else
        parentNode->right = newNode; // 오른쪽에 삽입
    return 1;
}

int deleteLeafNode(Node* head, int key)
{
    if (head == NULL) { //삭제할 것이 없을때
        printf("\n Nothing to delete!!\n");
        return -1;
    }

    if (head->left == NULL) { //삭제할 것이 없을때
        printf("\n Nothing to delete!!\n");
        return -1;
    }

    /* head->left is the root */
    Node* ptr = head->left;


    /* we have to move onto children nodes,
     * keep tracking the parent using parentNode */
    Node* parentNode = head;

    while(ptr != NULL) {

        if(ptr->key == key) { //삭제할 key 값을 찾았다면
            if(ptr->left == NULL && ptr->right == NULL) { //삭제할 노드의 자식노드가
존재하지 않는다면

                /* root node case */
                if(parentNode == head) //부모노드가 head 노드라면
                    head->left = NULL; // left 노드를 비워준다.

                /* left node case or right case*/
                if(parentNode->left == ptr) //부모노드의 왼쪽 노드가 삭제할 노드라면
                    parentNode->left = NULL; //왼쪽노드를 비워준다.
                else
                    parentNode->right = NULL; //오른쪽노드를 비워준다.

                free(ptr); //메모리해제
            }
            else {
```

```
                printf("the node [%d] is not a leaf \n", ptr->key); //리프노드가
아닐때 출력
            }
            return 1;
        }

        /* keep the parent node */
        parentNode = ptr; // 부모노드를 현재노드로 바꾸어줌.

        /* key comparison, if current node's key is greater than input key
         * then the new node has to be inserted into the right subtree;
         * otherwise the left subtree.
         */
        if(ptr->key < key) // 삽입된 key 값이 더 크면 오른쪽
            ptr = ptr->right; // 노드로 이동
        else
            ptr = ptr->left; // 왼쪽 노드로 이동


    }

    printf("Cannot find the node for key [%d]\n ", key); // 삭제할 key 를 찾지 못했을
때 출력

    return 1;
}

Node* searchRecursive(Node* ptr, int key)
{
    if(ptr == NULL)
        return NULL;

    if(ptr->key < key) //찾는 key 값이 ptr 의 Key 보다 크면
        ptr = searchRecursive(ptr->right, key); // 오른쪽 자식노드로 이동
    else if(ptr->key > key) //작으면
        ptr = searchRecursive(ptr->left, key); //왼쪽 자식노드로 이동

    /* if ptr->key == key */
    return ptr;

}
Node* searchIterative(Node* head, int key)
{
    /* root node */
    Node* ptr = head->left;

    while(ptr != NULL) // ptr 이 NULL 이 아닐때 까지
    {
```

```
        if(ptr->key == key) //key 값을 찾으면 return
            return ptr;

        if(ptr->key < key) ptr = ptr->right; //더 크다면 오른쪽 자식노드로 이동
        else //작다면 왼쪽 자식으로 이동
            ptr = ptr->left;
    }

    return NULL;
}

void freeNode(Node* ptr)
{
    if(ptr) { //Leaf 노드의 왼쪽노드부터 메모리 해제
        freeNode(ptr->left);
        freeNode(ptr->right);
        free(ptr);
    }
}

int freeBST(Node* head)
{

    if(head->left == head) //노드가 비어있다면
    {
        free(head); //head 노드의 메모리 해제
        return 1;
    }

    Node* p = head->left;

    freeNode(p); //전체 트리의 메모리 해제

    free(head); //head 노드의 메모리 해제
    return 1;
}
```

```
------------------------------------------------------------------
Command = s
Your Key = 4

 node [4] found at 0x13d104080


------------------------------------------------------------------
                   Binary Search Tree #1
------------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = n      Delete Leaf Node             = d
 Inorder Traversal    = i      Search Node Recursively      = s
 Preorder Traversal   = p      Search Node Iteratively      = f
 Postorder Traversal  = t      Quit                         = q
------------------------------------------------------------------
Command = i
 [1]  [2]  [3]  [4]

------------------------------------------------------------------
                   Binary Search Tree #1
------------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = n      Delete Leaf Node             = d
 Inorder Traversal    = i      Search Node Recursively      = s
 Preorder Traversal   = p      Search Node Iteratively      = f
 Postorder Traversal  = t      Quit                         = q
------------------------------------------------------------------
Command = p
 [1]  [2]  [3]  [4]

------------------------------------------------------------------
                   Binary Search Tree #1
------------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = n      Delete Leaf Node             = d
 Inorder Traversal    = i      Search Node Recursively      = s
 Preorder Traversal   = p      Search Node Iteratively      = f
 Postorder Traversal  = t      Quit                         = q
------------------------------------------------------------------
Command = t
 [4]  [3]  [2]  [1]

------------------------------------------------------------------
                   Binary Search Tree #1
------------------------------------------------------------------
 Initialize BST       = z
 Insert Node          = n      Delete Leaf Node             = d
 Inorder Traversal    = i      Search Node Recursively      = s
 Preorder Traversal   = p      Search Node Iteratively      = f
 Postorder Traversal  = t      Quit                         = q
------------------------------------------------------------------
Command = d
Your Key = 4
```

```
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 322 insertions(+)
 create mode 100644 binary-search-tree.c
➜  자료구조 git:(main) ✗ git push origin main
remote: Repository not found.
fatal: repository 'https://github.com/ohjaesik/homework9/' not found
➜  자료구조 git:(main) ✗ git push origin main
Enumerating objects: 129, done.
Counting objects: 100% (129/129), done.
Delta compression using up to 8 threads
Compressing objects: 100% (112/112), done.
Writing objects: 100% (129/129), 4.69 MiB | 7.03 MiB/s, done.
Total 129 (delta 44), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (44/44), done.
To https://github.com/ohjaesik/homework9
 * [new branch]      main -> main
➜  자료구조 git:(main) ✗ git rm --cached circuar-linked-list.c
fatal: pathspec 'circuar-linked-list.c' did not match any files
➜  자료구조 git:(main) ✗ git rm --cached circular-linked-list.c
rm 'circular-linked-list.c'
➜  자료구조 git:(main) ✗ git commit -m '삭제'
[main 6b4e34f] 삭제
 Committer: 오재식 <ojaesig@ojaesigs-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 354 deletions(-)
 delete mode 100644 circular-linked-list.c
➜  자료구조 git:(main) ✗ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (1/1), done.
Writing objects: 100% (2/2), 264 bytes | 264.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ohjaesik/homework9
   270df08..6b4e34f  main -> main
➜  자료구조 git:(main) ✗
```

https://github.com/ohjaesik/homework9